

1/3/1 (Item 1 from file: 351)
DIALOG(R)File 351:Derwent WPI
(c) 2006 The Thomson Corporation. All rts. reserv.

0013276065 - Drawing available
WPI ACC NO: 2003-362161/200334
XRPX Acc No: N2003-289206

Multi-thread execution method for parallel processing system, involves
determining forkability of slave thread and executing fork instructions of
slave thread or subsequent instruction of master thread

Patent Assignee: NEC CORP (NIDE)

Inventor: MATSUSHITA S; OHSAWA T; OSAWA H

Patent Family (5 patents, 3 countries)

Patent Number	Kind	Date	Application Number	Kind	Date	Update
US 20030014471	A1	20030116	US 2002133409	A	20020429	200334 B
JP 2003029984	A	20030131	JP 2001212246	A	20010712	200334 E
GB 2381610	A	20030507	GB 200216272	A	20020712	200338 E
JP 3702813	B2	20051005	JP 2001212246	A	20010712	200565 E
GB 2381610	B	20060222	GB 200216272	A	20020712	200615 E

Priority Applications (no., kind, date): JP 2001212246 A 20010712

Patent Details

Number	Kind	Lan	Pg	Dwg	Filing Notes
US 20030014471	A1	EN	40	24	
JP 2003029984	A	JA	21		
JP 3702813	B2	JA	28		Previously issued patent JP 2003029984

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2003-029984

(43)Date of publication of application : 31.01.2003

(51)Int.Cl.

G06F 9/46

G06F 15/16

G06F 15/177

(21)Application number : 2001-212246

(71)Applicant : NEC CORP

(22)Date of filing : 12.07.2001

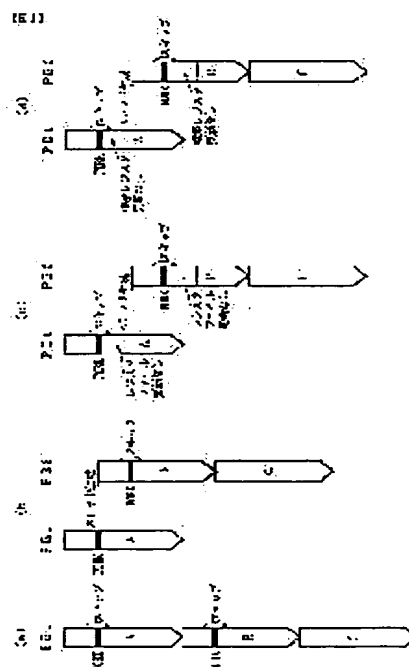
(72)Inventor : OSAWA HIROSHI
MATSUSHITA SATOSHI

(54) MULTITHREAD EXECUTION METHOD AND PARALLEL PROCESSOR SYSTEM

(57)Abstract:

PROBLEM TO BE SOLVED: To provide a multithread execution method executing the processing of a program without failure and without the interruption of the processing even in the case that a free processor capable of generating a slave thread is not present at the point in time of the fork instruction of a master thread while preventing the increase of the amount of hardware due to the provision of many register files and the increase of overheads at the time of the process changeover of an OS.

SOLUTION: At the time of dividing a single program into a plurality of threads A-C and parallelly executing them in a plurality of processors PE1 and PE2, the forking possibility of the slave thread B to the other processor PE2 by the fork instruction in the master thread A executed in the processor PE1 is judged and the slave thread B is forked to the processor PE2 when forking is possible. When forking is impossible, the fork instruction is invalidated, succeeding instructions after the fork instruction are continuously executed in the processor PE1 and then the instruction group of the slave thread B is executed in the processor PE1.



LEGAL STATUS

[Date of request for examination] 17.01.2002

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number] 3702813

[Date of registration] 29.07.2005

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's
decision of rejection]

[Date of extinction of right]

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号
特開2003-29984
(P2003-29984A)

(43) 公開日 平成15年1月31日 (2003.1.31)

(51) Int.Cl. ⁷	識別記号	F I	テームト* (参考)
G 0 6 F 9/46	3 6 0	G 0 6 F 9/46	3 6 0 B 5 B 0 4 5
15/16	6 1 0	15/16	6 1 0 Z 5 B 0 9 8
15/177	6 7 4	15/177	6 7 4 A
	6 8 1		6 8 1 B

審査請求 有 請求項の数22 O L (全 21 頁)

(21) 出願番号 特願2001-212246(P2001-212246)

(22) 出願日 平成13年7月12日 (2001.7.12)

(71) 出願人 000004237

日本電気株式会社

東京都港区芝五丁目7番1号

(72) 発明者 大澤 拓

東京都港区芝五丁目7番1号 日本電気株式会社内

(72) 発明者 松下 智

東京都港区芝五丁目7番1号 日本電気株式会社内

(74) 代理人 100088959

弁理士 境 廣巳

Fターム(参考) 5B045 GC11

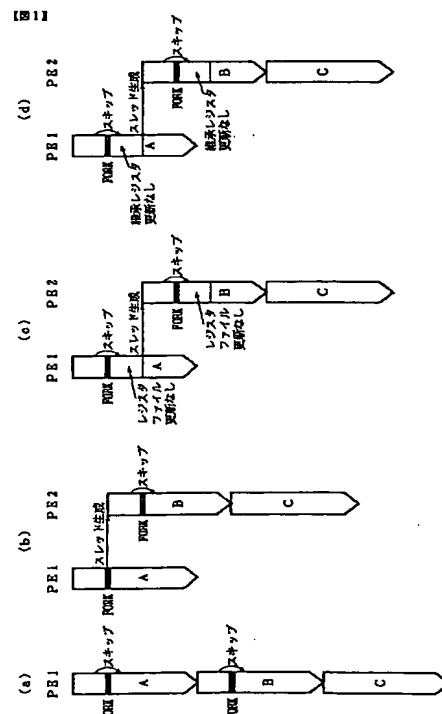
5B098 AA10 GA05 GC14

(54) 【発明の名称】 マルチスレッド実行方法及び並列プロセッサシステム

(57) 【要約】

【課題】 多数のレジスタファイルを持つことによるハードウェア量の増加、OSのプロセス切り替え時におけるオーバーヘッドの増大を防止しつつ、親スレッドのフォーク命令時点で子スレッドを生成できる空きのプロセッサが存在しない場合でも処理の中断無しにプログラムの処理を支障なく遂行できるマルチスレッド実行方法を提供する。

【解決手段】 単一のプログラムを複数のスレッドA～Cに分割し、複数のプロセッサPE1、PE2で並列に実行する際、プロセッサPE1で実行している親スレッドA中のフォーク命令による他プロセッサPE2への子スレッドBのフォーク可能性を判定し、フォーク可能ならば子スレッドBをプロセッサPE2にフォークし、フォーク不可能ならば、フォーク命令を無効化してプロセッサPE1でフォーク命令以降の後続命令を引き続き実行した後、子スレッドBの命令群をプロセッサPE1で実行する。



【特許請求の範囲】

【請求項 1】 単一のプログラムを複数のスレッドに分割し複数のプロセッサで並列に実行するマルチスレッド実行方法において、

親スレッド中のフォーク命令による他プロセッサへの子スレッドのフォーク可能性を判定し、フォーク可能ならば前記子スレッドをフォークし、フォーク不可能ならば前記親スレッドを実行しているプロセッサで前記フォーク命令以降の後続命令を引き続き実行した後に前記子スレッドの命令群を実行することを特徴とするマルチスレッド実行方法。

【請求項 2】 前記フォーク可能性の判定を前記フォーク命令の時点でのみ実施する請求項 1 記載のマルチスレッド実行方法。

【請求項 3】 前記フォーク可能性の判定は、前記フォーク命令の時点で前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって行う請求項 2 記載のマルチスレッド実行方法。

【請求項 4】 前記フォーク可能性の判定を前記フォーク命令の時点及びその時点でフォーク不可能と判定した場合には前記フォーク命令以降の時点でも実施する請求項 1 記載のマルチスレッド実行方法。

【請求項 5】 前記フォーク命令の時点での前記フォーク可能性の判定は、前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって行い、前記フォーク命令以降の時点での前記フォーク可能性の判定は、前記親スレッドのレジスタファイルが更新される前に前記子スレッドの実行を開始できる他のプロセッサが生じたか否かによって行う請求項 4 記載のマルチスレッド実行方法。

【請求項 6】 前記フォーク命令の時点での前記フォーク可能性の判定は、前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって行い、前記フォーク命令以降の時点での前記フォーク可能性の判定は、前記親スレッドのレジスタファイル中のレジスタのうち前記子スレッドに継承すべきレジスタが更新される前に前記子スレッドの実行を開始できる他のプロセッサが生じたか否かによって行う請求項 4 記載のマルチスレッド実行方法。

【請求項 7】 前記子スレッドのフォークが行われたときに前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットし、前記フォークドビットがセットされた前記プロセッサで実行中のスレッドのターム命令は有効化し、前記フォークドビットがセットされない前記プロセッサで実行中のスレッドのターム命令は無効化する請求項 1 乃至 6 の何れか 1 項に記載のマルチスレッド実行方法。

【請求項 8】 前記親スレッドのフォーク命令の時点でフォーク先アドレスを前記親スレッドを実行する前記プロセッサに設けたレジスタに保存し、前記子スレッドの

フォークが行われたときに前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットし、前記フォークドビットがセットされており且つプログラムカウンタの値が前記レジスタに保存された前記フォーク先アドレスと一致した前記プロセッサはスレッドの処理を終了する請求項 1 乃至 6 の何れか 1 項に記載のマルチスレッド実行方法。

【請求項 9】 各々プログラムカウンタ及びレジスタファイルを独立に有し前記プログラムカウンタに従ってスレッドの命令を同時にフェッチ、解釈、実行する複数のプロセッサを備え、何れかの前記プロセッサで実行されている親スレッド中のフォーク命令によって指定されたフォーク先アドレスから始まる子スレッドの実行を、前記フォーク命令時点の前記親スレッドのレジスタファイルのうち少なくとも前記子スレッドに必要なレジスタの値を前記子スレッドに継承させて、他の前記プロセッサに開始させる機能を備えた並列プロセッサシステムにおけるマルチスレッド実行方法において、(a) 親スレッド中のフォーク命令の時点で前記フォーク命令による他プロセッサへの子スレッドのフォーク可能性を、前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって判定するステップ、(b) フォーク可能ならば前記子スレッドをフォークして前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットし、フォーク不可能ならば前記フォーク命令を無効化し、前記親スレッドを実行しているプロセッサで前記フォーク命令以降の後続命令を引き続き実行するステップ、(c) 前記フォークドビットがセットされた前記プロセッサで実行中のスレッドのターム命令は有効化し、前記フォークドビットがセットされない前記プロセッサで実行中のスレッドのターム命令は無効化するステップ、を含むことを特徴とするマルチスレッド実行方法。

【請求項 10】 各々プログラムカウンタ及びレジスタファイルを独立に有し前記プログラムカウンタに従ってスレッドの命令を同時にフェッチ、解釈、実行する複数のプロセッサを備え、何れかの前記プロセッサで実行されている親スレッド中のフォーク命令によって指定されたフォーク先アドレスから始まる子スレッドの実行を、前記フォーク命令時点の前記親スレッドのレジスタファイルのうち少なくとも前記子スレッドに必要なレジスタの値を前記子スレッドに継承させて、他の前記プロセッサに開始させる機能を備えた並列プロセッサシステムにおけるマルチスレッド実行方法において、(a) 親スレッド中のフォーク命令の時点で、フォーク先アドレスを前記親スレッドを実行する前記プロセッサに設けたレジスタに保存すると共に、前記フォーク命令による他プロセッサへの子スレッドのフォーク可能性を、前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって判定するステップ、(b) フォーク可能ならば前記子スレッドをフォークして前記親スレッドを実行

する前記プロセッサに設けたフォークドビットをセットし、フォーク不可能ならば前記フォーク命令を無効化し、前記親スレッドを実行しているプロセッサで前記フォーク命令以降の後続命令を引き続き実行するステップ、(c) 前記フォークドビットがセットされており且つプログラムカウンタの値が前記レジスタに保存された前記フォーク先アドレスと一致した前記プロセッサはスレッドの処理を終了するステップ、を含むことを特徴とするマルチスレッド実行方法。

【請求項11】 各々プログラムカウンタ及びレジスタファイルを独立に有し前記プログラムカウンタに従ってスレッドの命令を同時にフェッチ、解釈、実行する複数のプロセッサを備え、何れかの前記プロセッサで実行されている親スレッド中のフォーク命令によって指定されたフォーク先アドレスから始まる子スレッドの実行を、前記フォーク命令時点の前記親スレッドのレジスタファイルのうち少なくとも前記子スレッドに必要なレジスタの値を前記子スレッドに継承させて、他の前記プロセッサに開始させる機能を備えた並列プロセッサシステムにおけるマルチスレッド実行方法において、(a) 親スレッド中のフォーク命令の時点で、フォーク先アドレスを前記親スレッドを実行する前記プロセッサに設けたレジスタに保存すると共に、前記フォーク命令による他プロセッサへの子スレッドのフォーク可能性を、前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって判定するステップ、(b) ステップaの判定がフォーク可能ならば前記子スレッドをフォークして前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットし、フォーク不可能ならば前記フォーク命令によるフォークを保留し、前記親スレッドを実行しているプロセッサで前記フォーク命令以降の後続命令を引き続き実行するステップ、(c) ステップbでフォークを保留した場合、前記フォーク命令以降の後続命令の実行と並行して、前記フォーク命令による他プロセッサへの子スレッドのフォーク可能性を、前記親スレッドのレジスタファイルの更新前に前記子スレッドの実行を開始できる他のプロセッサが生じたか否かによって判定し、フォーク可能と判定された時点で、前記レジスタに保存されたフォーク先アドレスに従って子スレッドをフォークして前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットするステップ、

(d) 前記フォークドビットがセットされた前記プロセッサで実行中のスレッドのターム命令は有効化し、前記フォークドビットがセットされない前記プロセッサで実行中のスレッドのターム命令は無効化するステップ、を含むことを特徴とするマルチスレッド実行方法。

【請求項12】 各々プログラムカウンタ及びレジスタファイルを独立に有し前記プログラムカウンタに従ってスレッドの命令を同時にフェッチ、解釈、実行する複数のプロセッサを備え、何れかの前記プロセッサで実行さ

れている親スレッド中のフォーク命令によって指定されたフォーク先アドレスから始まる子スレッドの実行を、前記フォーク命令時点の前記親スレッドのレジスタファイルのうち少なくとも前記子スレッドに必要なレジスタの値を前記子スレッドに継承させて、他の前記プロセッサに開始させる機能を備えた並列プロセッサシステムにおけるマルチスレッド実行方法において、(a) 親スレッド中のフォーク命令の時点で、フォーク先アドレスを前記親スレッドを実行する前記プロセッサに設けたレジスタに保存すると共に、前記フォーク命令による他プロセッサへの子スレッドのフォーク可能性を、前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって判定するステップ、(b) ステップaの判定がフォーク可能ならば前記子スレッドをフォークして前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットし、フォーク不可能ならば前記フォーク命令によるフォークを保留し、前記親スレッドを実行しているプロセッサで前記フォーク命令以降の後続命令を引き続き実行するステップ、(c) ステップaでフォーク不可能と判定された場合、前記フォーク命令以降の後続命令の実行と並行して、前記フォーク命令による他プロセッサへの子スレッドのフォーク可能性を、前記親スレッドのレジスタファイルの更新前に前記子スレッドの実行を開始できる他のプロセッサが生じたか否かによって判定し、フォーク可能となった時点で、前記レジスタに保存されたフォーク先アドレスに従って子スレッドをフォークして前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットするステップ、(d) 前記フォークドビットがセットされており且つプログラムカウンタの値が前記レジスタに保存された前記フォーク先アドレスと一致した前記プロセッサはスレッドの処理を終了するステップ、を含むことを特徴とするマルチスレッド実行方法。

【請求項13】 各々プログラムカウンタ及びレジスタファイルを独立に有し前記プログラムカウンタに従ってスレッドの命令を同時にフェッチ、解釈、実行する複数のプロセッサを備え、何れかの前記プロセッサで実行されている親スレッド中のフォーク命令によって指定されたフォーク先アドレスから始まる子スレッドの実行を、前記フォーク命令時点の前記親スレッドのレジスタファイルのうち少なくとも前記子スレッドに必要なレジスタの値を前記子スレッドに継承させて、他の前記プロセッサに開始させる機能を備えた並列プロセッサシステムにおけるマルチスレッド実行方法において、(a) 親スレッド中のフォーク命令の時点で、フォーク先アドレスを前記親スレッドを実行する前記プロセッサに設けたレジスタに保存すると共に、前記フォーク命令による他プロセッサへの子スレッドのフォーク可能性を、前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって判定するステップ、(b) ステップaの判定

がフォーク可能ならば前記子スレッドをフォークして前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットし、フォーク不可能ならば前記フォーク命令によるフォークを保留し、前記親スレッドを実行しているプロセッサで前記フォーク命令以降の後続命令を引き続き実行するステップ、(c) ステップ a でフォーク不可能と判定された場合、前記フォーク命令以降の後続命令の実行と並行して、前記フォーク命令による他プロセッサへの子スレッドのフォーク可能性を、前記親スレッドのレジスタファイル中のレジスタのうち前記子スレッドに継承すべきレジスタが更新される前に前記子スレッドの実行を開始できる他のプロセッサが生じたか否かによって判定し、フォーク可能と判定された時点で、前記レジスタに保存されたフォーク先アドレスに従って子スレッドをフォークして前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットするステップ、(d) 前記フォークドビットがセットされた前記プロセッサで実行中のスレッドのターム命令は有効化し、前記フォークドビットがセットされない前記プロセッサで実行中のスレッドのターム命令は無効化するステップ、を含むことを特徴とするマルチスレッド実行方法。

【請求項 14】 各々プログラムカウンタ及びレジスタファイルを独立に有し前記プログラムカウンタに従ってスレッドの命令を同時にフェッチ、解釈、実行する複数のプロセッサを備え、何れかの前記プロセッサで実行されている親スレッド中のフォーク命令によって指定されたフォーク先アドレスから始まる子スレッドの実行を、前記フォーク命令時点の前記親スレッドのレジスタファイルのうち少なくとも前記子スレッドに必要なレジスタの値を前記子スレッドに継承させて、他の前記プロセッサに開始させる機能を備えた並列プロセッサシステムにおけるマルチスレッド実行方法において、(a) 親スレッド中のフォーク命令の時点で、フォーク先アドレスを前記親スレッドを実行する前記プロセッサに設けたレジスタに保存すると共に、前記フォーク命令による他プロセッサへの子スレッドのフォーク可能性を、前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって判定するステップ、(b) ステップ a の判定がフォーク可能ならば前記子スレッドをフォークして前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットし、フォーク不可能ならば前記フォーク命令によるフォークを保留し、前記親スレッドを実行しているプロセッサで前記フォーク命令以降の後続命令を引き続き実行するステップ、(c) ステップ a でフォーク不可能と判定された場合、前記フォーク命令以降の後続命令の実行と並行して、前記フォーク命令による他プロセッサへの子スレッドのフォーク可能性を、前記親スレッドのレジスタファイル中のレジスタのうち前記子スレッドに継承すべきレジスタが更新される前に前記

子スレッドの実行を開始できる他のプロセッサが生じたか否かによって判定し、フォーク可能と判定された時点で、前記レジスタに保存されたフォーク先アドレスに従って子スレッドをフォークして前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットするステップ、(d) 前記フォークドビットがセットされており且つプログラムカウンタの値が前記レジスタに保存された前記フォーク先アドレスと一致した前記プロセッサはスレッドの処理を終了するステップ、を含むことを特徴とするマルチスレッド実行方法。

【請求項 15】 単一のプログラムを複数のスレッドに分割し複数のプロセッサで並列に実行する並列プロセッサシステムにおいて、親スレッド中のフォーク命令による他プロセッサへの子スレッドのフォーク可能性を判定する手段と、フォーク可能ならば前記子スレッドをフォークし、フォーク不可能ならば前記親スレッドを実行しているプロセッサで前記フォーク命令以降の後続命令を引き続き実行した後に前記子スレッドの命令群を実行する手段とを備えたことを特徴とする並列プロセッサシステム。

【請求項 16】 前記フォーク可能性の判定を前記フォーク命令の時点でのみ実施する構成を有する請求項 15 記載の並列プロセッサシステム。

【請求項 17】 前記フォーク可能性の判定は、前記フォーク命令の時点で前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって行う請求項 16 記載の並列プロセッサシステム。

【請求項 18】 前記フォーク可能性の判定を前記フォーク命令の時点及びその時点でフォーク不可能と判定した場合には前記フォーク命令以降の時点でも実施する構成を有する請求項 15 記載の並列プロセッサシステム。

【請求項 19】 前記フォーク命令の時点での前記フォーク可能性の判定は、前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって行い、前記フォーク命令以降の時点での前記フォーク可能性の判定は、前記親スレッドのレジスタファイルが更新される前に前記子スレッドの実行を開始できる他のプロセッサが生じたか否かによって行う請求項 18 記載の並列プロセッサシステム。

【請求項 20】 前記フォーク命令の時点での前記フォーク可能性の判定は、前記子スレッドの実行を開始できる他のプロセッサが存在するか否かによって行い、前記フォーク命令以降の時点での前記フォーク可能性の判定は、前記親スレッドのレジスタファイル中のレジスタのうち前記子スレッドに継承すべきレジスタが更新される前に前記子スレッドの実行を開始できる他のプロセッサが生じたか否かによって行う請求項 18 記載の並列プロセッサシステム。

【請求項 21】 前記子スレッドのフォークが行われたときに前記親スレッドを実行する前記プロセッサに設け

たフォークドビットをセットする手段と、前記フォークドビットがセットされた前記プロセッサで実行中のスレッドのターム命令は有効化し、前記フォークドビットがセットされない前記プロセッサで実行中のスレッドのターム命令は無効化する手段とを備えた請求項 15 乃至 20 の何れか 1 項に記載の並列プロセッサシステム。

【請求項 22】 前記親スレッドのフォーク命令の時点でフォーク先アドレスを前記親スレッドを実行する前記プロセッサに設けたレジスタに保存する手段と、前記子スレッドのフォークが行われたときに前記親スレッドを実行する前記プロセッサに設けたフォークドビットをセットする手段とを備え、前記フォークドビットがセットされており且つプログラムカウンタの値が前記レジスタに保存された前記フォーク先アドレスと一致した前記プロセッサはスレッドの処理を終了する構成を有する請求項 15 乃至 20 の何れか 1 項に記載の並列プロセッサシステム。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】 本発明は並列プロセッサシステムにおけるプログラム並列実行方法に関し、より具体的には単一のプログラムを複数のスレッドに分割して複数のプロセッサにより並列に実行するマルチスレッド実行方法及び並列プロセッサシステムに関する。

【0002】

【従来の技術】 単一のプログラムを並列プロセッサシステムで並列に処理する手法として、プログラムをスレッドと呼ぶ命令流に分割して複数のプロセッサで並列に実行するマルチスレッド実行方法があり、この方法を記載した文献として、特開平 10-27108 号公報（以下、文献 1 と称す）、「On Chip Multiprocessor 指向 制御並列アーキテクチャ MUSCAT の提案」（並列処理シンポジウム JSP97 論文集、情報処理学会、pp. 229-236、May 1997）（以下、文献 2 と称す）、特開平 10-78880 号公報（以下、文献 3 と称す）等がある。以下、これらの従来文献に記載されたマルチスレッド実行方法について説明する。

【0003】 一般にマルチスレッド実行方法において、他のプロセッサ上に新たなスレッドを生成することを、スレッドをフォーク（fork）すると言い、フォーク動作を行った側のスレッドを親スレッド、生成された新しいスレッドを子スレッド、スレッドをフォークする箇所をフォーク点、子スレッドの先頭箇所をフォーク先アドレスまたは子スレッドの開始点と呼ぶ。文献 1～3 では、スレッドのフォークを指示するためにフォーク点にフォーク命令が挿入される。フォーク命令にはフォーク先アドレスが指定され、フォーク命令の実行によりそのフォーク先アドレスから始まる子スレッドが他プロセッサ上に生成され、子スレッドの実行が開始される。ま

た、スレッドの処理を終了させるターム（term）命令と呼ばれる命令が用意されており、各プロセッサはターム命令を実行することによりスレッドの処理を終了する。

【0004】 図 22 に従来のマルチスレッド実行方法の処理の概要を示す。同図（a）は 3 つのスレッド A、B、C に分割された単一のプログラムを示す。このプログラムを単一のプロセッサで処理する場合、同図（b）に示すように 1 つのプロセッサ PE がスレッド A、B、C を順番に処理していく。これに対して文献 1～3 のマルチスレッド実行方法では、同図（c）に示すように、1 つのプロセッサ PE1 にスレッド A を実行させ、プロセッサ PE1 でスレッド A を実行している最中に、スレッド A に埋め込まれたフォーク命令によってスレッド B を他のプロセッサ PE2 に生成し、プロセッサ PE2 においてスレッド B を実行させる。また、プロセッサ PE2 はスレッド B に埋め込まれたフォーク命令によってスレッド C をプロセッサ PE3 に生成する。プロセッサ PE1、PE2 はそれぞれスレッド B、C の開始点の直前に埋め込まれたターム命令によってスレッドの処理を終了し、プロセッサ PE3 はスレッド C の最後の命令を実行すると、その次の命令（一般にはシステムコール命令）を実行する。このように複数のプロセッサでスレッドを同時に並行して実行することにより、逐次処理に比べて性能の向上が図られる。

【0005】 従来の他のマルチスレッド実行方法として、図 22（d）に示すように、スレッド A を実行しているプロセッサ PE1 からフォークを複数回行うことにより、プロセッサ PE2 にスレッド B を、またプロセッサ PE3 にスレッド C をそれぞれ生成するマルチスレッド実行方法も存在する。この図 22（d）のモデルに対して、図 22（c）に示したようにスレッドはその生存中に高々 1 回に限って有効な子スレッドを生成することができるという制約を課したマルチスレッド実行方法をフォーク 1 回モデルと呼ぶ。フォーク 1 回モデルでは、スレッド管理の大幅な簡略化が可能となり、現実的なハードウェア規模でスレッド管理部のハードウェア化が実現できる。本発明はこのようなフォーク 1 回モデルを前提とする。

【0006】 ここで、フォーク命令時、子スレッドを生成できる空きのプロセッサが存在しない場合、従来は次の 2 通りの方法の何れかを採用している。

（1）親スレッドを実行しているプロセッサは、子スレッドを生成できる空きのプロセッサが生じるまで、フォーク命令の実行をウェイトする。

（2）親スレッドを実行しているプロセッサは、フォーク点におけるレジスタファイルの内容（フォーク先アドレス及びレジスタ内容）を裏面の物理レジスタに保存して親スレッドの後続処理を続行する。裏面の物理レジスタに保存されたレジスタファイルの内容は、子スレッド

を生成できる空きのプロセッサが生じた時点で参照され、子スレッドが生成される。

【0007】親スレッドが子スレッドを生成し、子スレッドに所定の処理を行わせるには、親スレッドのフォーク点におけるレジスタファイル中のレジスタのうち少なくとも子スレッドに必要なレジスタの値を親スレッドから子スレッドに引き渡す必要がある。このスレッド間のデータ引き渡しコストを削減するために、文献2及び3では、スレッド生成時のレジスタ値継承機構をハードウェア的に備えている。これは、スレッド生成時に親スレッドのレジスタファイルの内容を子スレッドに全てコピーするものである。子スレッド生成後は、親スレッドと子スレッドのレジスタ値の変更は独立となり、レジスタを用いたスレッド間のデータの引き渡しは行われない。スレッド間のデータ引き渡しに関する他の従来技術としては、レジスタの値を命令によりレジスタ単位で個別に転送する機構を備えた並列プロセッサシステムも提案されている。

【0008】その他、文献2に記載のMUSCATでは、スレッド間の同期命令など、スレッドの並列動作を柔軟に制御するための専用命令が数多く用意されている。

【0009】

【発明が解決しようとする課題】上述したように従来のマルチスレッド実行方法においては、子スレッドを生成できる空きのプロセッサが存在しない場合、空きのプロセッサが生じるまで親スレッドのフォーク命令の実行をウェイトしているため、場合によっては長時間待たされ、処理効率が極端に低下するという課題がある。

【0010】この処理停止による処理効率の低下を改善するため、フォーク点におけるレジスタファイルの内容を裏面の物理レジスタに保存して親スレッドの処理の続行を可能にする方法では、各プロセッサ毎に表裏の少なくとも2面のレジスタファイルが必要になる。1つのレジスタファイルに例えば32ビットのレジスタが32個収納されているとすると、1プロセッサ当たり32×32ビットのメモリが必要になり、1チップにn個のプロセッサを集積化したオンチップ並列プロセッサでは無視できないハードウェア量の増加となる。また、単にハードウェア量の増加だけでなく、オペレーティングシステム(OS)によるプロセス切り替え時には裏面の物理レジスタも表の物理レジスタと一緒に退避、復元する必要があるため、プロセス切り替え時の処理量が增大し、プロセス切り替え時のオーバーヘッドの増大による性能の低下を招く。

【0011】また上述した従来のマルチスレッド実行方法においては、スレッドを終了させるためには必ずターム命令を子スレッドの開始点の直前に記述しておく必要がある。ターム命令は1スレッド当たり1個必要になるため、1つのスレッドに含まれる命令数が少ない細粒度

スレッドほど、全命令数に占めるターム命令の割合が多くなる。ターム命令も他の命令と同様に命令メモリに格納されてフェッチの対象となるため、命令メモリのハードウェア量の増加、命令フェッチ数の増加による処理性能の低下が問題となる。

【0012】本発明はこのような従来の問題点を解決したものであり、その目的は、多数のレジスタファイルを持つことによるハードウェア量の増加、OSのプロセス切り替え時におけるオーバーヘッドの増大を防止しつつ、親スレッドのフォーク命令時点で子スレッドを生成できる空きのプロセッサが存在しない場合でも処理の中断無しにプログラムの処理を支障なく遂行できる新規なマルチスレッド実行方法及び並列プロセッサシステムを提供することにある。

【0013】また本発明の別の目的は、スレッドを終了させるためのターム命令を削減することにより命令メモリに必要な容量を削減し、また命令フェッチ数の削減による処理性能の向上を図ることにある。

【0014】

【課題を解決するための手段】本発明は、単一のプログラムを複数のスレッドに分割し複数のプロセッサで並列に実行する際、親スレッド中のフォーク命令による他プロセッサへの子スレッドのフォーク可能性を判定し、フォーク可能ならば子スレッドをフォークし、フォーク不可能ならば親スレッドを実行しているプロセッサでフォーク命令以降の後続命令を引き続き実行した後に子スレッドの命令群を実行することを基本とする。

【0015】より具体的には、第1の発明は、親スレッド中のフォーク命令による子スレッドのフォーク可能性の判定を、子スレッドの実行を開始できる他のプロセッサが存在するか否かによって、フォーク命令の時点でのみ実施し、可能ならばフォークし、その時点でフォーク不可能であれば当該子スレッドのフォークは断念して、子スレッドの命令群は親スレッドを実行しているプロセッサで実行する。

【0016】また、第2の発明は、フォーク可能性の判定を子スレッドの実行を開始できる他のプロセッサが存在するか否かによってフォーク命令の時点で判定し、可能ならばフォークし、その時点でフォーク不可能であってもフォークを即断念せずに保留し、フォーク命令以降の後続命令の処理と並行して、親スレッドのレジスタファイルが更新される前に子スレッドの実行を開始できる他のプロセッサが生じたか否かによってフォーク可能性の判定を続け、フォーク可能となった時点でフォークし、最終的にフォーク不可能ならば当該子スレッドのフォークは断念して、子スレッドの命令群は親スレッドを実行しているプロセッサで実行する。

【0017】また、第3の発明は、フォーク可能性の判定を子スレッドの実行を開始できる他のプロセッサが存在するか否かによってフォーク命令の時点で判定し、可

能ならばフォークし、その時点でフォーク不可能であってもフォークは即断念せずに保留し、フォーク命令以降の後続命令の処理と並行して、親スレッドのレジスタファイル中のレジスタのうち子スレッドに継承すべきレジスタが更新される前に子スレッドの実行を開始できる他のプロセッサが生じたか否かによってフォーク可能性の判定を続け、フォーク可能となった時点でフォークし、最終的にフォーク不可能ならば当該子スレッドのフォークは断念して、子スレッドの命令群は親スレッドを実行しているプロセッサで実行する。

【0018】また、第4の発明は、子スレッドのフォークが行われたときに親スレッドを実行するプロセッサに設けたフォークドビットをセットし、フォークドビットがセットされたプロセッサで実行中のスレッドのターム命令は無効化し、フォークドビットがセットされないプロセッサで実行中のスレッドのターム命令は無効化する。

【0019】また、第5の発明は、親スレッドのフォーク命令の時点でフォーク先アドレスを親スレッドを実行するプロセッサに設けたレジスタに保存し、子スレッドのフォークが行われたときに親スレッドを実行するプロセッサに設けたフォークドビットをセットし、フォークドビットがセットされており且つプログラムカウンタの値がレジスタに保存されたフォーク先アドレスと一致したプロセッサはスレッドの処理を終了する。

【0020】

【作用】本発明にあつては、親スレッド中のフォーク命令による他プロセッサへの子スレッドのフォーク可能性を判定し、フォーク不可能ならば、親スレッドを実行しているプロセッサで後続命令を引き続き実行するため、処理が中断することがなく、また、親スレッドの後続命令を実行した後に子スレッドの命令群を実行するため、プログラムの処理を支障なく遂行でき、更に、子スレッドの処理を親スレッドと同じプロセッサで行うため、裏面のレジスタファイルにフォーク点のレジスタファイルの内容を退避しておく必要がなくなり、多数のレジスタファイルを持つことによるハードウェア量の増加、OSのプロセス切り替え時におけるオーバヘッドの増大を防止できる。

【0021】以下、図22(a)に示した3つのスレッドA、B、Cに分割された単一のプログラムを例に本発明の作用を説明する。

【0022】(1) 第1の発明

図1(a)に示すように、1つのプロセッサPE1にスレッドAを実行させ、プロセッサPE1でスレッドAを実行しているときにフォーク命令に実行が差しかかると、子スレッドBの実行を開始することができる他プロセッサが存在するかが判定される。図1(a)では、そのようなプロセッサが存在しないため、当該フォーク命令は無効化され、後続命令の処理が引き続き実行

され、且つ、スレッドAの処理後に引き続きスレッドBの処理が開始される。同様に、図1(a)ではスレッドBのフォーク命令の実行が不可能であるため、そのフォーク命令は無効化されて後続命令の処理が引き続き実行され、その後にスレッドCの処理が実行されている。この場合のスレッドの実行シーケンスは、A→B→Cであり、図22(b)で説明した単一のプロセッサによる逐次実行順序と同じであり、プログラムの処理は支障なく遂行できる。

10 【0023】図1(b)は、スレッドAからスレッドBはフォークできたが、スレッドBからスレッドCはフォークできなかった場合の実行シーケンスを示す。この場合、スレッドAを実行していたプロセッサPE1はスレッドBをフォークしたので、スレッドAだけを実行し、プロセッサPE2はスレッドCをフォークできなかったため、スレッドBに引き続きスレッドCを実行する。

20 【0024】第1の発明では、フォーク命令の時点で子スレッドの実行を開始できる他プロセッサが存在していないと最早フォークは行えないが、第2及び第3の発明のようなレジスタファイルの更新を考慮したフォーク可能性の判定が不要になるため、制御の簡素化およびハードウェア量の削減が可能である。

【0025】(2) 第2の発明

図1(c)に示すように、1つのプロセッサPE1にスレッドAを実行させ、プロセッサPE1でスレッドAを実行しているときにフォーク命令に実行が差しかかると、子スレッドBの実行を開始することができる他プロセッサが存在するかが判定される。図1(c)では、そのようなプロセッサが存在しないため、当該フォーク命令は保留され、プロセッサPE1は後続命令の処理を続行する。そして、スレッドAのレジスタファイルが更新される前に子スレッドBの実行を開始することができる他プロセッサPE2が生じたため、フォーク可能と判定され、プロセッサPE2にスレッドBが生成されて実行が開始される。プロセッサPE2ではスレッドB中のフォーク命令に実行が差しかかると、子スレッドCの実行を開始することができる他プロセッサが存在するかが判定される。図1(c)では、そのようなプロセッサが存在しないため、当該フォーク命令は保留され、プロセッサPE2は後続命令の処理を続行する。また、図1(c)では、スレッドBのレジスタファイルが更新される前に子スレッドCの実行を開始することができる他プロセッサが生じなかったため、フォークを断念し、プロセッサPE2はスレッドBの処理を終了すると、引き続きスレッドCの処理を実行している。

50 【0026】また、プロセッサPE1において、スレッドAのレジスタファイルが更新される前に子スレッドBの実行を開始することができる他プロセッサが存在しなかった場合はスレッドAに引き続きスレッドBを実行し、またスレッドBの実行中に、スレッドBのレジスタ

ファイルが更新される前に子スレッドCの実行を開始することができる他プロセッサが存在しなかった場合はスレッドBに引き続きスレッドCも実行する。この際のスレッドの実行シーケンスは図1(a)と同じになる。

【0027】この第2の発明では、フォーク命令の時点でフォークできなくとも、レジスタファイルが更新される前に子スレッドの実行を開始できる他プロセッサが生じるとフォークを行うため、第1の発明に比べてフォークされる可能性が高まり、スレッド実行の並列度が向上する。

【0028】(3) 第3の発明

図1(d)に示すように、1つのプロセッサPE1にスレッドAを実行させ、プロセッサPE1でスレッドAを実行しているときにフォーク命令に実行が差しかかると、子スレッドBの実行を開始することができる他プロセッサが存在するか否かが判定される。図1(d)では、そのようなプロセッサが存在しないため、当該フォーク命令は保留され、プロセッサPE1は後続命令の処理を続行する。そして、図1(d)では、スレッドAのレジスタファイルのうち子スレッドBに継承すべきレジスタの値が更新される前に子スレッドBの実行を開始することができる他プロセッサPE2が生じたため、フォーク可能と判定され、プロセッサPE2にスレッドBが生成されて実行が開始される。プロセッサPE2ではスレッドB中のフォーク命令に実行が差しかかると、子スレッドCの実行を開始することができる他プロセッサが存在するか否かが判定される。図1(d)では、そのようなプロセッサが存在しないため、当該フォーク命令は保留され、プロセッサPE2は後続命令の処理を続行する。また、図1(d)では、スレッドBのレジスタファイル中の子スレッドCに継承すべきレジスタの値が更新される前に子スレッドCの実行を開始することができる他プロセッサが生じなかったため、フォークを断念し、プロセッサPE2はスレッドBの処理を終了すると、引き続きスレッドCの処理を実行している。

【0029】また、プロセッサPE1において、スレッドAのレジスタファイル中の子スレッドBに継承すべきレジスタの値が更新される前に子スレッドBの実行を開始することができる他プロセッサが存在しなかった場合はスレッドAに引き続きスレッドBを実行し、またスレッドBの実行中に、スレッドBのレジスタファイル中の子スレッドCに継承すべきレジスタの値が更新される前に子スレッドCの実行を開始することができる他プロセッサが存在しなかった場合はスレッドBに引き続きスレッドCも実行する。この際のスレッドの実行シーケンスは図1(a)と同じになる。

【0030】この第3の発明では、親スレッドのレジスタファイルの更新があっても、その更新が子スレッドに継承すべきレジスタでなければフォークを行うため、第2の発明に比べてフォークされる可能性をより高めるこ

とができ、従ってスレッド実行の並列度をより向上することができる。

【0031】(4) 第4の発明

この第4の発明では、従来と同様に子スレッドの開始点の直前にターム命令が置かれる。図1(b)、(c)、(d)において、プロセッサPE1では親スレッドAから子スレッドBをフォークしたのでフォークドビットがセットされる。他方、プロセッサPE2では親スレッドBから子スレッドCをフォークしなかったためフォークドビットはセットされない。このため、プロセッサPE1では、子スレッドBの開始点の直前のターム命令は有効になり、親スレッドAの実行によりスレッドの処理を終了する。他方、プロセッサPE2では、子スレッドCの開始点の直前のターム命令は無効化され、これによりスレッドの処理を停止することなく親スレッドBの実行後に子スレッドCの命令群の実行が開始される。

【0032】(5) 第5の発明

この第5の発明では、従来と異なり子スレッドの開始点の直前にはターム命令は存在しない。図1(b)、(c)、(d)において、プロセッサPE1はスレッドAのフォーク命令の時点でフォーク先アドレス(スレッドBの開始アドレス)をレジスタに保存し、親スレッドAから子スレッドBをフォークしたのでフォークドビットがセットされる。他方、プロセッサPE2はスレッドBのフォーク命令の時点でフォーク先アドレス(スレッドCの開始アドレス)をレジスタに保存するが、親スレッドBから子スレッドCをフォークしなかったためフォークドビットはセットされない。このため、プロセッサPE1では、プログラムカウンタの値がレジスタに保存されたフォーク先アドレス(スレッドBの開始アドレス)と一致した時点でスレッドの処理を終了する。他方、プロセッサPE2では、プログラムカウンタの値がレジスタに保存されたフォーク先アドレス(スレッドBの開始アドレス)と一致してもスレッドの処理は終了せず、子スレッドCの命令群の処理へと進む。

【0033】なお、本発明においては、親スレッドから子スレッドへのフォーク時におけるレジスタの値の継承は、フォーク命令時点の親スレッドのレジスタファイルのうち少なくとも子スレッドに必要なレジスタだけを対象とすれば足りる。このための具体的なレジスタ継承機構としては、文献2及び文献3に記載されるようにスレッド生成時に親スレッドのレジスタファイルの内容すべてを子スレッドのレジスタファイルにコピーするものであっても良いし、レジスタ転送量の削減を図るために必要なレジスタの値だけを命令によりレジスタ単位で個別に転送するものであっても良い。

【0034】

【発明の実施の形態】次に本発明の実施の形態の例について図面を参照して詳細に説明する。

【0035】

【第1の実施の形態】図2を参照すると、本発明の並列プロセッサシステムの一例は、4スレッド並列実行型プロセッサであり、4個のプロセッサ1-i (i=0~3) が信号線2-iによってスレッド管理部3に接続されると共に、信号線4-iによって共有のメモリ5に接続されている。また、プロセッサ1-i相互間は通信バス6で接続されている。この例では、4スレッド並列実行型プロセッサを取り上げたが、8スレッドや16スレッドの並列実行型プロセッサ等、一般にn (≧2) スレッド並列実行型プロセッサに対して本発明は適用可能である。全てのプロセッサ1-i、メモリ5及びスレッド管理部3はクロックに同期して動作する。また、好ましくは、全てのプロセッサ1-iはメモリ5及びスレッド管理部3と共に1つの半導体チップ上に集積化される。

【0036】各プロセッサ1-iは、プログラムカウンタ(以下、PCと称す)及びレジスタファイルを独立に有し、PCに従って、メモリ5中のスレッドの命令を同時にフェッチ、解釈、実行する機能を有している。各プロセッサ1-iにおけるスレッドの実行は、スレッド管理部3から信号線2-iを通じてターゲットPC値を伴うスレッド開始要求7cが送信された時点で開始される。スレッドの実行を終了したプロセッサ1-iは、スレッド管理部3に対して信号線2-iを通じてスレッド終了通知7dを送信する。このスレッド終了通知7dがスレッド管理部3で受理された時点で、当該プロセッサ1-iはフリー状態として管理され、新たなスレッドの実行を当該プロセッサ1-iに開始させることができる。

【0037】各プロセッサ1-iは、実行中の親スレッドに存在するフォーク命令によって他のプロセッサ1-j (i≠j) に子スレッドをフォークすることができる。その際、プロセッサ1-iは、信号線2-iを通じてスレッド管理部3に対し、子スレッドのフォーク先アドレス(開始PC値)を伴うフォーク要求7aを送信する。スレッド管理部3は、フォーク要求7aを受信すると、子スレッドの実行を開始できる他のプロセッサ1-jが存在するか否かを調べ、存在すれば当該他のプロセッサ1-jに対してフォーク先アドレスを伴うスレッド開始要求7cを送信する一方、フォーク要求元のプロセッサ1-iに対しては、当該他のプロセッサ1-jの番号を指定したフォーク応答7bを返却する。この時点で初めてフォークが行われたことになり、フォーク応答7bを受信したプロセッサ1-iは、フォーク先のプロセッサ1-jのレジスタファイルに対して、親スレッドのレジスタファイルの全内容を通信バス6を通じてコピーするか、当該子スレッドに必要なレジスタの値だけをコピーすることにより、レジスタ継承を行う。

【0038】他方、子スレッドの実行を開始できる他のプロセッサが存在しなかった場合、スレッド管理部3は、今回のフォーク要求7aを廃棄する。これにより前

記フォーク命令は無効化される。

【0039】図3を参照すると、スレッド管理部3の一例は、スレッド管理シーケンサ11とプロセッサ状態テーブル12とから構成される。プロセッサ状態テーブル12は、プロセッサ1-iと1対1に対応するエントリ13-iを有する。個々のエントリ13-iは、対応するプロセッサ1-iがビジー状態か、フリー状態かを記録するために使用される。スレッド管理シーケンサ11は、このプロセッサ状態テーブル12を用いて各プロセッサ1-iにおけるスレッド生成、スレッド終了を管理する。プロセッサ1-iからフォーク要求7a、スレッド終了通知7dを受信した際のスレッド管理シーケンサ11の処理例を図4及び図5に示す。

【0040】図4を参照すると、スレッド管理シーケンサ11は、或るクロックのタイミングで何れかのプロセッサ1-iからフォーク要求7aを受信すると、子スレッドの実行を開始できるプロセッサが存在するか否かをプロセッサ状態テーブル12を参照して調べる(ステップS1)。文献1~3に記載されるように、スレッド管理の簡便化のためにプロセッサ1-iから子スレッドをフォークできるプロセッサを、プロセッサ1-iの一方の隣接プロセッサ(プロセッサ1-0はプロセッサ1-1、プロセッサ1-1はプロセッサ1-2、プロセッサ1-2はプロセッサ1-3、プロセッサ1-3はプロセッサ1-0)に限定したモデル(このようなモデルを以下、リング型フォークモデルと称す)では、プロセッサ状態テーブル12におけるプロセッサ1-iに隣接するプロセッサ1-jに対応するエントリ13-jを参照し、フリー状態であれば子スレッドの実行を開始できるプロセッサが存在すると判定でき、ビジー状態であればそのようなプロセッサは存在しないと判定できる。

【0041】子スレッドの実行を開始できるプロセッサ1-jが存在した場合、スレッド管理シーケンサ11は、プロセッサ状態テーブル12における当該プロセッサ1-jに対応するエントリ13-jをフリー状態からビジー状態に更新し(ステップS2)、フォーク要求7aに付随するフォーク先アドレスを添えたスレッド開始要求7cをフォーク先プロセッサ1-jに送信すると共に、要求元のプロセッサ1-iに対してフォーク先プロセッサ1-jを指定したフォーク応答7bを返却する

(ステップS3)。リング型フォークモデルでは、フォーク先プロセッサは事前に特定されるので、フォーク応答7bでフォーク先プロセッサ1-jを指定する必要はない。

【0042】他方、子スレッドの実行を開始できるプロセッサ1-jが存在しなかった場合、スレッド管理シーケンサ11は、当該フォーク要求7aを破棄する(ステップS4)。

【0043】図5を参照すると、スレッド管理シーケンサ11は、何れかのプロセッサ1-iからスレッド終了

通知 7 d を受信すると、プロセッサ管理テーブル 12 における当該プロセッサ 1-i に対応するエントリ 13-i をビジー状態からフリー状態に更新する（ステップ S 11）。

【0044】図 6 を参照すると、各々のプロセッサ 1-i は、スレッド管理部 3 から送信されたスレッド開始要求 7 c に付随する開始アドレス値がセットされ、その後適宜歩進される PC 21 と、PC 21 に従ってメモリ 5 からスレッドの命令をフェッチする命令フェッチユニット 22 と、フェッチされた命令をデコードし、実行する実行ユニット 23 と、汎用レジスタ 24-0 ~ 24-m の集合であるレジスタファイル 25 と、フォーク先プロセッサに対して通信バス 6 経由でレジスタファイル 25 の内容を転送するレジスタ転送ユニット 26 と、フォーク命令実行時に実行ユニット 23 からスレッド管理部 3 に送信されたフォーク先アドレスを伴うフォーク要求 7 a に対するフォーク応答 7 b によってセットされるフォークドビット 27 とを含んで構成され、フォークドビット 27 の値は実行ユニット 23 に入力されている。

【0045】実行ユニット 23 は、スレッド中のターム命令のデコード時、フォークドビット 27 がセットされているか否かを判別し、セットされているときは当該ターム命令を有効に実行することによりスレッドの処理を終了する。この際、スレッド管理部 3 に対してスレッド終了通知 7 d を送信する。他方、フォークドビット 27 がセットされていないときは当該ターム命令を無効化し、PC 21 に従って後続命令の処理を続行する。また、レジスタ転送ユニット 26 は、フォークドビット 27 がセットされるタイミングでフォーク先プロセッサへのレジスタ転送を開始する。レジスタ転送ユニット 26 は、例えば、通信バス 6 のバス幅によって一度に転送できる数のレジスタ毎に、レジスタファイル 25 のレジスタの値とレジスタ番号（レジスタアドレス）とをフォーク先プロセッサのレジスタファイルへ送信し、受信側のレジスタファイル 25 では該当するレジスタを書き換える。

【0046】次に本実施の形態にかかるマルチスレッド実行方法の動作を、スレッドの開始から終了までのプロセッサ 1-i 及びスレッド管理部 3 の処理の一例を示す図 7 のフローチャートを参照して説明する。

【0047】スレッド管理部 3 からのスレッド開始要求 7 c に基づき、プロセッサ 1-i で 1 つのスレッドの実行が開始される際、当該プロセッサ 1-i のフォークドビット 27 がリセットされ（ステップ S 21）、スレッドの命令のフェッチ、デコード、実行が以後継続して実行される（ステップ S 22）。図 7 では、スレッド中の命令のうち、ターム命令とフォーク命令について特に注目して処理の概要を例示してある。

【0048】実行ユニット 23 でデコードされた命令がフォーク命令の場合（ステップ S 24 で YES）、実行

ユニット 23 はフォーク先アドレスを指定したフォーク要求 7 a をスレッド管理部 3 に送信し、スレッド管理部 3 は前述したようにして子スレッドの実行を開始できるフリー状態の他プロセッサ 1-j が存在するか否かを調べる（ステップ S 25）。フリー状態の他プロセッサ 1-j が存在した場合、スレッド管理部 3 は前述したようにプロセッサ 1-i のフォークドビット 27 をセットすると同時に他プロセッサ 1-j に対してスレッド開始要求 7 c を送出することで子スレッドをフォークする（ステップ S 26）。この際、プロセッサ 1-i のレジスタ転送ユニット 26 は、レジスタファイル 25 の内容を通信バス 6 経由でプロセッサ 1-j へ送信し、プロセッサ 1-j のレジスタファイル 25 に書き込む。他方、子スレッドの実行を開始できる他プロセッサが存在しなかった場合、スレッド管理部 3 はプロセッサ 1-i から送信されたフォーク要求 7 a を廃棄することでフォーク命令を無効化する。プロセッサ 1-i ではフォーク命令の実行後、PC 21 に従って次の命令の実行を継続する（ステップ S 22）。

【0049】実行ユニット 23 でデコードされた命令がターム命令の場合（ステップ S 23 で YES）、実行ユニット 23 はフォークドビット 27 がセットされていれば（ステップ S 27 で YES）、当該ターム命令を実行することによりスレッドの処理を終了する（ステップ S 28）。しかし、フォークドビット 27 がセットされていなければ、当該ターム命令を無効化し、PC 21 に従って次の命令の実行を継続する（ステップ S 22）。

【0050】リング型フォークモデルに適用した本実施の形態にかかるマルチスレッド実行方法の実行シーケンスの一例を図 8 に示す。図 8 では、プロセッサ #0 からその隣接プロセッサ #1 に、プロセッサ #1 からその隣接プロセッサ #2 に、プロセッサ #2 からその隣接プロセッサ #3 にそれぞれ子スレッドをフォークしている。また、プロセッサ #3 のフォーク点でプロセッサ #0 がフリー状態となっているため、プロセッサ #3 からプロセッサ #0 への子スレッドのフォークも成功している。しかし、このプロセッサ #0 に新たに生成されたスレッドのフォーク点では、隣接プロセッサ #1 がビジー状態であるためフォークは不可能であり、フォークはスキップ（無効化）されている。このため、プロセッサ #0 は当該スレッドのターム命令をスキップ（無効化）し、本来は隣接プロセッサ #1 で行うべき子スレッドを自身で実行している。

【0051】次に、本実施の形態にかかるマルチスレッド実行方法で実行される並列化プログラムの生成方法について説明する。

【0052】図 9 を参照すると、コンパイラ 41 は、逐次処理プログラム 42 を入力し、制御及びデータフロー解析部 44 によって逐次処理プログラム 42 の制御フロー及びデータフローを解析して、基本ブロック或いは複

数の基本ブロックを並列化の単位、すなわちスレッドに分割し、次いで並列化コード挿入部 45 によって並列化のためのコードを挿入して、複数のスレッドに分割された並列化プログラム 43 を生成して出力する。本発明では、並列化プログラム 43 中の各スレッドはその生存中に高々 1 回に限って有効な子スレッドを生成するというフォーク 1 回制限を並列化プログラム 43 において静的に保証している。

【0053】並列化コードとしては、フォーク命令、ターム命令などがある。本実施の形態では並列化プログラムの生成時に、フォーク点にフォーク命令が挿入され、且つ子スレッドの開始点の直前にターム命令が挿入される。また、前述したように本実施の形態では、フォーク不可能ならばフォーク命令が無効化され、フォーク命令が無効化されたときには対応するターム命令も無効化されるため、コンパイラ 41 は、フォーク命令、ターム命令がたとえ無効化されても逐次処理プログラム 42 と等価な処理が行える並列化プログラム 43 を生成する。一般に、並列化プログラム 43 中のフォーク命令及びターム命令を全て取り除いた状態の制御フローが逐次処理プログラム 42 の制御フローと等価であれば、逐次処理プログラム 42 の動作を保証できる並列化プログラム 43 となる。

【0054】図 10 (a) に逐次処理プログラム 42 の一例を示す。この例の逐次処理プログラム 42 は、米国 MIPS Technology INC. 社の RISC プロセッサの命令セットを用いて記述されており、レジスタ r14 に 5 を加算し (addu)、レジスタ r16 の指し示すアドレスにレジスタ r14 の内容をストアし (sw)、レジスタ r16 の指し示すアドレス + 4 の内容をレジスタ r1 にロードし (lw)、ループを実行するよう記述されている。このループでは、__func を開始アドレスとする関数呼び出しを行い (jal)、レジスタ r1 の値を -1 し (sub)、レジスタ r1 とレジスタ r0 (常に 0) を比較し等しくなければ loop へ分岐する (bne)、処理を記述してある。

【0055】図 10 (b) に図 10 (a) の逐次処理プログラム 42 をコンパイラ 41 がコンパイルして生成した並列化プログラム 43 の一例を示す。この例では、addu 命令の直前に、__th1 をフォーク先アドレスとする 1 つのフォーク命令 (fork) が挿入され、このフォーク先アドレス __th1 の直前にターム命令 (term) が挿入されている。また、この子スレッド中に __th2 をフォーク先アドレスとする 1 つのフォーク命令が挿入され、このフォーク先アドレス __th2 の直前にターム命令が挿入されている。

【0056】図 10 (b) の並列化プログラム 43 からフォーク命令及びターム命令を全て除去すると、図 10 (a) の逐次処理プログラム 42 と同じ制御フローのプログラムとなり、フォーク命令、ターム命令が無効化さ

れても逐次処理プログラム 42 で遂行される処理が保証される並列化プログラムとなっているのが分かる。

【0057】なお、図 10 (b) の並列化プログラム 43 には、親スレッドのレジスタファイル中のレジスタのうち、子スレッドに継承すべきレジスタの情報が含まれていない為、子スレッドのフォーク時には親スレッドのレジスタファイルの内容を全て転送することになる。勿論、後述する第 3 の実施の形態と同様に子スレッドに継承すべきレジスタをコンパイル時に解析して並列化プログラム 43 に継承レジスタの情報を記述しておき、子スレッドのフォーク時に親スレッドのレジスタファイルの内容のうち子スレッドで必要なレジスタだけを転送するように構成することも可能である。

【0058】

【第 2 の実施の形態】本実施の形態は、並列化プログラム中のターム命令を不要にした点で第 1 の実施の形態と相違する。以下、第 1 の実施の形態との相違点を中心に本実施の形態を説明する。

【0059】図 11 を参照すると、本実施の形態における並列プロセッサシステムの各々のプロセッサ 1-i は、図 6 に示した構成に加えて、実行ユニット 23 からスレッド管理部 3 に送信されるスレッド要求 7a に付随するフォーク先アドレスを保存するレジスタ 28 と、PC21 の値がレジスタ 28 に保存されたフォーク先アドレスと一致するか否かを判定する一致回路 29 と、フォークドビット 27 及び一致回路 29 の出力の論理積出力を実行ユニット 23 に出力するアンドゲート 30 とを含んで構成されている。

【0060】実行ユニット 23 は、フォークドビット 27 がセットされている状態において PC21 の値がレジスタ 28 に保存されたフォーク先アドレスと一致することによりアンドゲート 30 の出力が論理 “1” になると、スレッドの処理を終了し、スレッド管理部 3 に対してスレッド終了通知 7d を送信する。PC21 の値がレジスタ 28 に保存されたフォーク先アドレスと一致しても、フォークドビット 27 がセットされていないければ、アンドゲート 30 の出力は論理 “1” にならないため、実行ユニット 23 は PC21 に従って命令の実行を継続する。

【0061】本実施の形態におけるスレッドの開始から終了までのプロセッサ 1-i 及びスレッド管理部 3 の処理の一例を図 12 に示す。図 7 との相違点はステップ S31、S32 である。

【0062】実行ユニット 23 でデコードされた命令がフォーク命令の場合 (ステップ S24 で YES)、実行ユニット 23 はフォーク命令で指定されたフォーク先アドレスをレジスタ 28 に保存し (ステップ S32)、このレジスタ 28 の出力を伴ってフォーク要求 7a がスレッド管理部 3 に送信される。スレッド管理部 3 は第 1 の実施の形態と同様に子スレッドの実行を開始できるフリ

一状態の他プロセッサ 1-j が存在すれば（ステップ S 25 で YES）、プロセッサ 1-i のフォークドビット 27 をセットすると同時に他プロセッサ 1-j に対してスレッド開始要求 7c を送出することで子スレッドをフォークする（ステップ S 26）。他方、子スレッドの実行を開始できる他プロセッサが存在しなかった場合、スレッド管理部 3 はプロセッサ 1-i から送信されたフォーク要求 7a を廃棄することでフォーク命令を無効化する。

【0063】プロセッサ 1-i で命令の実行が進み、PC 21 の値がレジスタ 28 に保存されたフォーク先アドレスに一致すると（ステップ S 31 で YES）、フォークドビット 27 がセットされていれば（ステップ S 27 で YES）、アンドゲート 30 の出力が論理“1”となり、実行ユニット 23 に割り込みがかかり、当該プロセッサ 1-i はスレッドの処理を終了する（ステップ S 28）。しかし、フォークドビット 27 がセットされていなければ、PC 21 に従って命令の実行、つまり子スレッド命令群を継続して実行することになる（ステップ S 22）。

【0064】上述したように本実施の形態にかかるマルチスレッド実行方法では、並列化プログラム中のターム命令が不要になるため、図 9 のコンパイラ 41 における並列化コード挿入部 45 は、子スレッドの開始点の直前にターム命令は挿入しない。図 13 に、図 10 (a) の逐次処理プログラム 42 を本実施の形態にかかるマルチスレッド実行方法向けに生成した並列化プログラム 43 の一例を示す。図 10 (b) の並列化プログラムの子スレッドの開始点（-th1、-th2）の直前に挿入されていたターム命令は、図 13 の並列化プログラムでは省略されている。

【0065】

【第 3 の実施の形態】第 1 及び第 2 の実施の形態では、親スレッドのフォーク点でフォーク可能でなければフォークを即断念したが、本実施の形態では、親スレッドのレジスタファイルが更新される前に子スレッドの実行を開始できる他プロセッサが生じるとフォークを行う。以下、第 2 の実施の形態との相違点を中心に本実施の形態を説明する。

【0066】図 14 を参照すると、本実施の形態における並列プロセッサシステムの各々のプロセッサ 1-i は、図 11 に示した構成に加えて、フォーク有効ビット 31 を含んで構成されている。フォーク有効ビット 31 は、実行ユニット 23 がフォーク命令を実行したときに出力するフォーク信号 37 でセットされ、スレッド管理部 3 から受信されるフォーク応答 7b 及び実行ユニット 23 が親スレッドのレジスタファイル 25 中の何れかのレジスタを更新したときに出力するレジスタ更新信号 33 によってリセットされる。フォーク有効ビット 31 の出力がスレッド管理部 3 に対するフォーク要求 7a とな

り、フォーク有効ビット 31 がセットされている間、フォーク要求 7a が送出し続けられる。

【0067】前述の図 4 を参照すると、スレッド管理部 3 のスレッド管理シーケンサ 11 は、或るクロックのタイミングでプロセッサ 1-i からフォーク要求 7a を受信した際、子スレッドの実行を開始できるプロセッサが存在しないとき（ステップ S 1 で NO）、当該フォーク要求 7a は破棄したが、前述したように、プロセッサ 1-i はフォーク有効ビット 31 がセットされている間、フォーク要求 7a を送出し続けているので、次のクロックのタイミングでスレッド管理部 3 がプロセッサ 1-i からフォーク要求 7a を再び受信することになり、図 4 の処理が繰り返される。即ち、フォーク点でフォーク不可能な場合、フォーク命令は保留にされ、フォーク可能となった時点で実行されることになる。

【0068】次に本実施の形態にかかるマルチスレッド実行方法の動作を、スレッドの開始から終了までのプロセッサ 1-i 及びスレッド管理部 3 の処理の概要を示す図 15 のフローチャートを参照して説明する。

【0069】スレッド管理部 3 からのスレッド開始要求 7c に基づき、プロセッサ 1-i で 1 つのスレッドの実行が開始される際、当該プロセッサ 1-i のフォーク有効ビット 31 及びフォークドビット 27 がリセットされ（ステップ S 41）、スレッドの命令のフェッチ、デコード、実行が以後継続して実行される（ステップ S 42）。図 15 では、スレッドの終了に関する処理とフォーク命令の処理について特にその概要を例示してある。

【0070】実行ユニット 23 でデコードされた命令がフォーク命令の場合（ステップ S 44 で YES）、実行ユニット 23 はフォーク先アドレスをレジスタ 28 に保存すると共にフォーク信号 37 を出力してフォーク有効ビット 31 をセットする（ステップ S 45）。また、実行ユニット 23 はレジスタファイル 25 中の何れかのレジスタを更新すると（ステップ S 46 で YES）、レジスタ更新信号 33 を出力してフォーク有効ビット 31 をリセットする（ステップ S 47）。従って、プロセッサ 1-i からは、フォーク命令実行時点からレジスタファイル 25 が最初に更新される迄の期間中、フォーク要求 7a がスレッド管理部 3 に送出し続けられる。

【0071】スレッド管理部 3 は、或るクロックのタイミングでプロセッサ 1-i からフォーク要求 7a を受信すると、図 4 に示したように、子スレッドの実行を開始できるフリー状態の他プロセッサ 1-j が存在するか否かを調べ（ステップ S 1）、フリー状態の他プロセッサ 1-j が存在した場合にはプロセッサ状態テーブル 12 を前述したように更新し（ステップ S 2）、プロセッサ 1-i にフォーク応答 7b を送信すると同時に他プロセッサ 1-j に対してスレッド開始要求 7c を送出することで子スレッドをフォークする（ステップ S 3）。プロセッサ 1-i に出されたフォーク応答 7b によって、フ

フォークドビット27がセットされ、フォーク有効ビット31はリセットされる。他方、子スレッドの実行を開始できる他プロセッサが存在しなかった場合（ステップS1でNO）、スレッド管理部3は今回のプロセッサ1-iから送信されたフォーク要求7aを破棄して図4の処理を終了するが、前述したようにプロセッサ1-iからはフォーク要求7aが送出し続けられている。図15のステップS51～S53は以上のような処理を別の観点でフローチャート化したものであり、プロセッサ1-iのフォーク有効ビット31がセットされ且つフリー状態のプロセッサ1-jが存在していれば、子スレッドのフォークを行い、フォーク要求元プロセッサ1-iのフォークドビット27をセットし且つフォーク有効ビット31をリセットすることを示している。なお、プロセッサ1-iでは、レジスタ転送ユニット26によるレジスタファイル25の転送中、実行ユニット23からレジスタファイル25への書き込みは待たされる。

【0072】なお、第2の実施の形態と同様に、プロセッサ1-iで命令の実行が進み、PC21の値がレジスタ28に保存されたフォーク先アドレスに一致すると（ステップS43でYES）、フォークドビット27がセットされていれば（ステップS48でYES）、アンドゲート30の出力が論理“1”となり、実行ユニット23に割り込みがかかって当該プロセッサ1-iはスレッドの処理を終了する（ステップS49）。

【0073】リング型フォークモデルに適用した本実施の形態にかかるマルチスレッド実行方法の実行シーケンスの一例を図16に示す。図8とほぼ同じ状況を想定しており、プロセッサ#3からプロセッサ#0にフォークされたスレッドのフォーク点Aでは、隣接プロセッサ#1はビジー状態である。このような状況の場合、第1及び第2の実施の形態ではフォークを即断念したが、本実施の形態ではフォーク点Aでフォーク先アドレスをレジスタ28に保存し、フォークを保留状態とする。フォーク点Aから下に延びる矢印は、プロセッサ#0においてレジスタファイル25が全く更新されていない期間を示す。図16では、この期間内にプロセッサ#1がフリー状態になったため、保留されたフォークが実行され、プロセッサ#1に子スレッドが生成されている。また、プロセッサ#0は子スレッドを生成したため、フォーク先アドレスに到達するとスレッドの処理を終了している。

【0074】なお、図14に示される各プロセッサ1-iの実行ユニット23が、ターム命令のデコード時、アンドゲート30の出力が論理“1”か否かを判別し、論理“1”ならば当該ターム命令を実行することによりスレッドの処理を終了し、論理“1”でなければ当該ターム命令を無効化し、PC21に従って後続命令の処理を続行するように構成すれば、第1の実施の形態と同様に、子スレッドの開始点の直前にターム命令が挿入された並列化プログラムを支障なく実施する別の実施の形態

が得られる。

【0075】

【第4の実施の形態】第3の実施の形態では、親スレッドのフォーク点でフォーク可能でなければフォークを一旦保留にし、親スレッドのレジスタファイルが更新される前に子スレッドの実行を開始できる他プロセッサが生じなかった場合に当該フォークを断念したが、本実施の形態では、親スレッドのレジスタファイルが更新されても、その更新が子スレッドに継承すべきレジスタでなければフォークを行う。以下、第3の実施の形態との相違点を中心に本実施の形態を説明する。

【0076】図17を参照すると、本実施の形態における並列プロセッサシステムの各々のプロセッサ1-iは、図14に示した構成に加えて、レジスタファイル25の各レジスタ24-k（k=0～m）に1対1に対応し、対応するレジスタ24-kが子スレッドへ継承すべきレジスタであるときに限りセットされるクリエイティブビット32-kと、各レジスタ24-kに1対1に対応し、対応するレジスタ24-kのクリエイティブビット32-kの出力と実行ユニット23がレジスタ24-kを更新したときに出力するレジスタ更新信号33-kとを入力とするアンドゲート34-kと、アンドゲート33-kの出力の論理和信号であるフォーク無効信号35を出力するオアゲート36とを含んで構成されている。そして、図14のレジスタ更新信号33に代えて、フォーク無効信号35がフォーク有効ビット31にリセット信号として出力されている。また、各クリエイティブビット32-kの値がレジスタ転送ユニット26に出力されており、レジスタ転送ユニット26はレジスタファイル25のレジスタ24-kのうち、対応するクリエイティブビット32-kがセットされているレジスタのみをフォーク先プロセッサのレジスタファイルに転送するように構成されている。

【0077】本実施の形態におけるスレッドの開始から終了までのプロセッサ1-i及びスレッド管理部3の処理の概要を図18に示す。図15との相違点はステップS61、S62である。

【0078】実行ユニット23でデコードされた命令がフォーク命令の場合（ステップS44でYES）、実行ユニット23はフォーク先アドレスをレジスタ28に保存すると共にフォーク信号37を出力してフォーク有効ビット31をセットし、且つ全てのクリエイティブビット32-kのセットアップを行う（ステップS61）。即ち、レジスタファイル25のレジスタ24-kのうち、子スレッドに継承すべきレジスタに対応するクリエイティブビット32-kはセットし、継承する必要のないレジスタに対応するクリエイティブビット32-kはリセットされたままにする。また、実行ユニット23はレジスタファイル25中の何れかのレジスタ24-kを更新すると、その更新したレジスタ24-kに対応するレジスタ更新

信号 33-k を論理 “1” とする。これにより、若し更新されたレジスタ 24-k が子スレッドへ継承すべきレジスタであった場合、そのレジスタ 24-k に対応するクリエイティブット 32-k はセットされているため、そのレジスタ 24-k に対応するアンドゲート 34-k の出力が論理 “1” となり、オアゲート 36 からフォーク無効信号 35 が出力されてフォーク有効ビット 31 がリセットされる（ステップ S62、S47）。つまり、プロセッサ 1-i からは、フォーク命令実行時点からレジスタファイル 25 中の子スレッドへの継承レジスタの何れかが最初に更新される迄の期間中、フォーク要求 7a がスレッド管理部 3 に送出し続けられる。

【0079】スレッド管理部 3 は、第 3 の実施の形態と同様の処理を行う。これによって、プロセッサ 1-i のレジスタファイル 24 のレジスタ 24-k のうち、子スレッドへ継承すべきレジスタが更新される前に子スレッドの実行を開始できる他プロセッサが生じると、プロセッサ 1-i にフォーク応答 7b を送信すると同時に他プロセッサ 1-j に対してスレッド開始要求 7c を送出することで子スレッドをフォークする（ステップ S51～53）。フォーク応答 7b を受信したプロセッサ 1-i のレジスタ転送ユニット 26 は、レジスタファイル 25 のレジスタ 24-k のうち、対応するクリエイティブット 32-k がセットされているレジスタのみをフォーク先プロセッサのレジスタファイルへ転送する。

【0080】なお、第 3 の実施の形態と同様に、プロセッサ 1-i で命令の実行が進み、PC21 の値がレジスタ 28 に保存されたフォーク先アドレスに一致すると（ステップ S43 で YES）、フォークドビット 27 がセットされていれば（ステップ S48 で YES）、アンドゲート 30 の出力が論理 “1” となり、実行ユニット 23 に割り込みがかかって当該プロセッサ 1-i はスレッドの処理を終了する（ステップ S49）。

【0081】リング型フォークモデルに適用した本実施の形態にかかるマルチスレッド実行方法の実行シーケンス例は図 16 と同じようになる。但し、フォーク点 A から下に延びる矢印は、本実施の形態ではプロセッサ #0 においてレジスタファイル 25 のうち子スレッドに継承すべきレジスタが全く更新されていない期間となり、その分だけフォーク可能期間が延長される。

【0082】本実施の形態では、親スレッドのフォーク点で子スレッドへ継承すべきレジスタが判明している必要がある。このため、図 9 に示したコンパイラ 41 における制御及びデータフロー解析部 44 では、フォークする子スレッド毎に、親スレッドから子スレッドへ継承すべきレジスタを調査し、並列化コード挿入部 45 ではその調査結果に基づいて、子スレッドへ継承すべきレジスタを指定する記述を並列化プログラム 43 に挿入する。

【0083】図 19 に、図 10 (a) の逐次処理プログラム 42 を本実施の形態にかかるマルチスレッド実行方

法向けに生成した並列化プログラム 43 の一例を示す。1 行目のフォーク命令中の「r1, r16, sp」、5 行目のフォーク命令中の「r1, sp」がそれぞれ子スレッドへ継承すべきレジスタの指定記述である。各プログラム 1-i の実行ユニット 23 はフォーク命令のデコード時、このような継承レジスタの指定を解釈し、指定されたレジスタ 24-k に対応するクリエイティブット 32-k のみをセットする。

【0084】図 20 に、図 10 (a) の逐次処理プログラム 42 を本実施の形態にかかるマルチスレッド実行方法向けに生成した並列化プログラム 43 の別の例を示す。図 19 のようにフォーク命令で継承レジスタを指定する方法では、フォーク命令の命令幅が大きくなるが、本例では、クリエイティブット (create) 命令という特殊命令を定義し、このクリエイティブット命令で子スレッドへ継承すべきレジスタを指定するため、フォーク命令の命令幅の増大が抑えられる。但し、クリエイティブット命令という特殊命令が追加されるため命令数は増加する。このため、クリエイティブット命令が存在しない場合には全レジスタ継承（或いはシステムで事前に設定された所定のレジスタ群の継承）としておき、クリエイティブット命令が存在すればそれで指定されたレジスタだけを継承するものとして扱う。従って、図 20 の 1 行目のフォーク命令にはその直前にクリエイティブット命令が存在しないので、全レジスタ継承と認識され、6 行目のフォーク命令にはその直前にレジスタ r1、sp を指定するクリエイティブット命令が存在するので、レジスタ r1、sp だけが継承対象となる。

【0085】図 21 に、図 10 (a) の逐次処理プログラム 42 を本実施の形態にかかるマルチスレッド実行方法向けに生成した並列化プログラム 43 の更に別の例を示す。図 20 ではフォーク命令の直前にクリエイティブット命令を挿入したが、本例ではフォーク命令の直後にクリエイティブット命令を挿入している。フォーク命令の直前にクリエイティブット命令を挿入すると、逐次動作で実行すべき命令が 1 命令増えるが、フォーク命令の直後にクリエイティブット命令を挿入するとフォーク命令をなるべく早く実行することができることによって並列に動作する部分を増やすことができる。

【0086】本実施の形態におけるレジスタ転送ユニット 26 は、クリエイティブット 32-k を参照することにより、親スレッドのレジスタファイル 25 のうち子スレッドに継承すべきレジスタだけをフォーク先プロセッサのレジスタファイルに転送するようにしたが、別の実施例として、レジスタファイル 25 の先頭のレジスタから順に所定の順番でレジスタの転送を行うシーケンスを開始し、クリエイティブット 32-k がセットされているレジスタの全ての転送が完了した時点で転送シーケンスを停止するようにしても良い。この方法では、子スレッドに継承する必要のないレジスタも転送される場合があるが、転送シーケンスが簡素化される利点がある。勿論、

別の実施例として、クリエイティブビット 32-k を一切参照せずに常に全レジスタを転送するようにレジスタ転送ユニット 26 が構成されていても良い。更に、子スレッドに継承すべきレジスタでも、フォーク先プロセッサの当該レジスタの値がフォーク時点で既に親スレッド側と同じ値になっている場合にはあえて転送する必要がない点に着目して、子スレッドに継承すべきレジスタのうち、親スレッド側と異なる値になっているレジスタを検出し、この検出したレジスタだけをレジスタ転送ユニット 26 からフォーク先プロセッサに転送するようにしても良い。

【0087】なお、図 17 に示される各プロセッサ 1-i の実行ユニット 23 が、ターム命令のデコード時、アンドゲート 30 の出力が論理“1”か否かを判別し、論理“1”ならば当該ターム命令を実行することによりスレッドの処理を終了し、論理“1”でなければ当該ターム命令を無効化し、PC 21 に従って後続命令の処理を続行するように構成すれば、第 1 の実施の形態と同様に、子スレッドの開始点の直前にターム命令が挿入された並列化プログラムを支障なく実施する別の実施の形態が得られる。

【0088】以上、本発明を幾つかの実施の形態を挙げて説明したが、本発明は以上の実施の形態にのみ限定されず、その他各種の付加変更が可能である。例えば、前記各実施の形態では、複数のプロセッサに共通にスレッド管理部 3 を設ける集中スレッド管理型の並列プロセッサシステムに本発明を適用したが、文献 1 等に記載されるように各プロセッサ毎にスレッド管理部を設ける分散スレッド管理型の並列プロセッサシステムにも本発明は適用可能である。また、プロセッサ相互間を通信バス 6 によって接続したが、リング型フォークモデルにあっては隣接するプロセッサ間どうしをリング上に通信線で接続する形態の並列プロセッサシステムに対しても本発明は適用可能である。

【0089】

【発明の効果】以上説明したように本発明によれば、多数のレジスタファイルを持つことによるハードウェア量の増加、OS のプロセス切り替え時におけるオーバヘッドの増大を防止しつつ、親スレッドのフォーク命令時点で子スレッドを生成できる空きのプロセッサが存在しない場合でも処理の中断無しにプログラムの処理を支障なく遂行することができる効果がある。

【0090】また第 2 の発明によれば、フォーク命令の時点でフォークできなくても、レジスタファイルが更新される前に子スレッドの実行を開始できる他プロセッサが生じるとフォークが可能になるため、第 1 の発明に比べてフォークできる確率が高まり、スレッド実行の並列度を向上することができる。

【0091】また第 3 の発明によれば、親スレッドのレジスタファイルの更新があっても、その更新が子スレ

ッドに継承すべきレジスタでなければフォークを行うため、第 2 の発明に比べてフォークできる確率を高めることができ、スレッド実行の並列度をより一層向上することができる。

【0092】また第 4 の発明によれば、従来と同様に子スレッドの開始点の直前にターム命令が置かれたプログラムを支障なく実行することが可能となる。

【0093】また第 5 の発明によれば、子スレッドの開始点の直前にターム命令を置く必要がなくなり、ターム命令の削減によってプログラムサイズをコンパクト化でき、命令メモリに必要な容量の削減、命令フェッチ数の削減による処理性能の向上が可能となる。

【図面の簡単な説明】

【図 1】本発明の作用の説明図である。

【図 2】本発明の並列プロセッサシステムの一例を示すブロック図である。

【図 3】本発明の並列プロセッサシステムにおけるスレッド管理部の構成例を示すブロック図である。

【図 4】本発明の並列プロセッサシステムにおけるスレッド管理部のスレッド管理シーケンサがプロセッサからフォーク要求を受信した際の処理例を示すフローチャートである。

【図 5】本発明の並列プロセッサシステムにおけるスレッド管理部のスレッド管理シーケンサがプロセッサからスレッド終了通知を受信した際の処理例を示すフローチャートである。

【図 6】本発明の並列プロセッサシステムにおけるプロセッサの構成例を示すブロック図である。

【図 7】本発明の並列プロセッサシステムにおけるスレッドの開始から終了までのプロセッサ及びスレッド管理部の処理の一例を示すフローチャートである。

【図 8】リング型フォークモデルに適用した本発明のマルチスレッド実行方法の実行シーケンスの一例を示す図である。

【図 9】本発明のマルチスレッド実行方法向けの並列化プログラムを生成するコンパイラの構成例を示すブロック図である。

【図 10】逐次処理プログラムとそれから生成された並列化プログラムの一例を示す図である。

【図 11】本発明の並列プロセッサシステムにおけるプロセッサの別の構成例を示すブロック図である。

【図 12】本発明の並列プロセッサシステムにおけるスレッドの開始から終了までのプロセッサ及びスレッド管理部の処理の他の例を示すフローチャートである。

【図 13】逐次処理プログラムから生成された並列化プログラムの他の例を示す図である。

【図 14】本発明の並列プロセッサシステムにおけるプロセッサの更に別の構成例を示すブロック図である。

【図 15】本発明の並列プロセッサシステムにおけるスレッドの開始から終了までのプロセッサ及びスレッド管

10

20

30

40

50

理部の処理の更に別の例を示すフローチャートである。

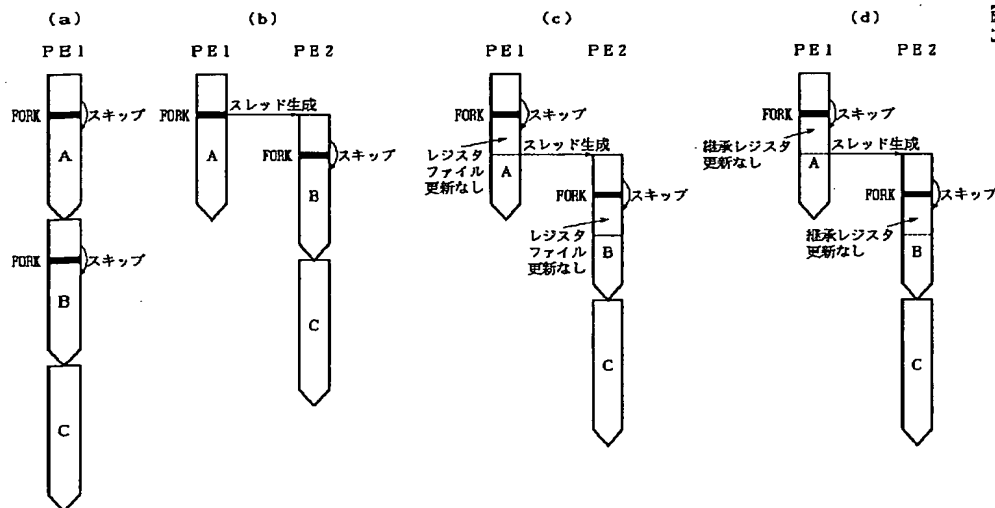
【図 16】リング型フォークモデルに適用した本発明のマルチスレッド実行方法の実行シーケンスの別の例を示す図である。

【図 17】本発明の並列プロセッサシステムにおけるプロセッサのまた更に別の構成例を示すブロック図である。

【図 18】本発明の並列プロセッサシステムにおけるスレッドの開始から終了までのプロセッサ及びスレッド管理部の処理のまた更に別の例を示すフローチャートである。

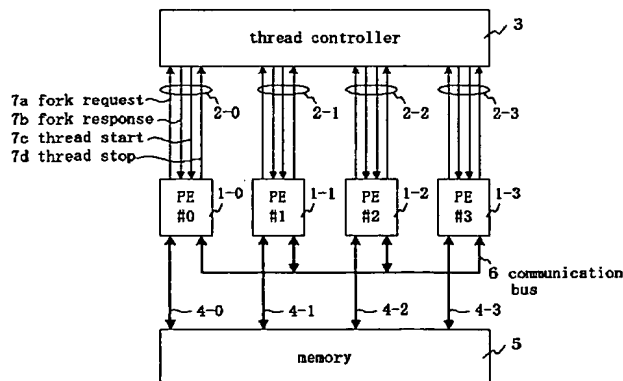
【図 19】逐次処理プログラムから生成された並列化プログラムの更に別の例を示す図である。

【図 1】



【図 2】

【図 2】



【図 20】逐次処理プログラムから生成された並列化プログラムのまた更に別の例を示す図である。

【図 21】逐次処理プログラムから生成された並列化プログラムの他の例を示す図である。

【図 22】従来のマルチスレッド実行方法の処理の概要を示す図である。

【符号の説明】

1-0~1-3...プロセッサ

2-0~2-3...信号線

3...スレッド管理部

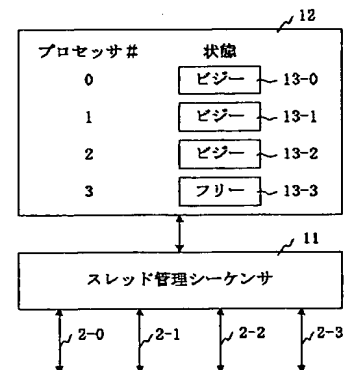
4-0~4-3...信号線

5...メモリ

6...通信バス

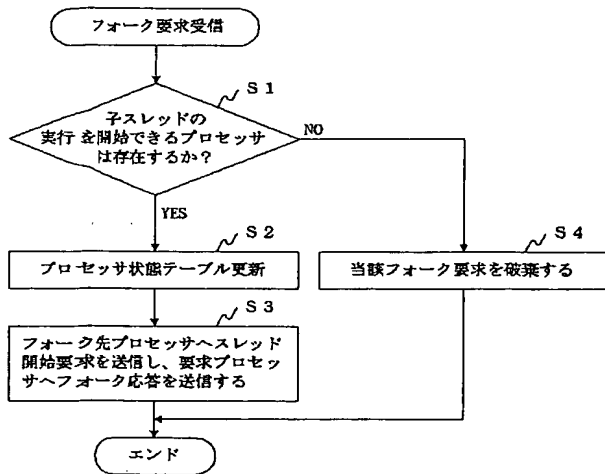
【図 3】

【図 3】



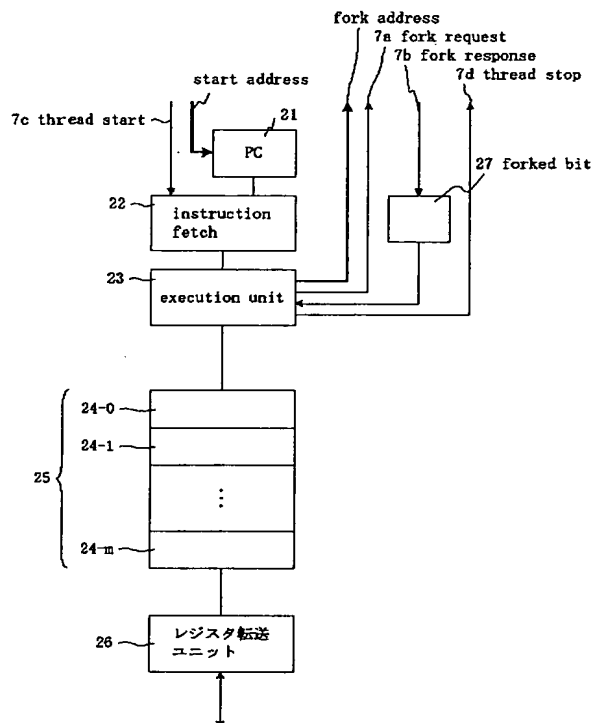
【図4】

【図4】



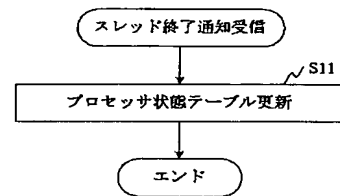
【図6】

【図6】



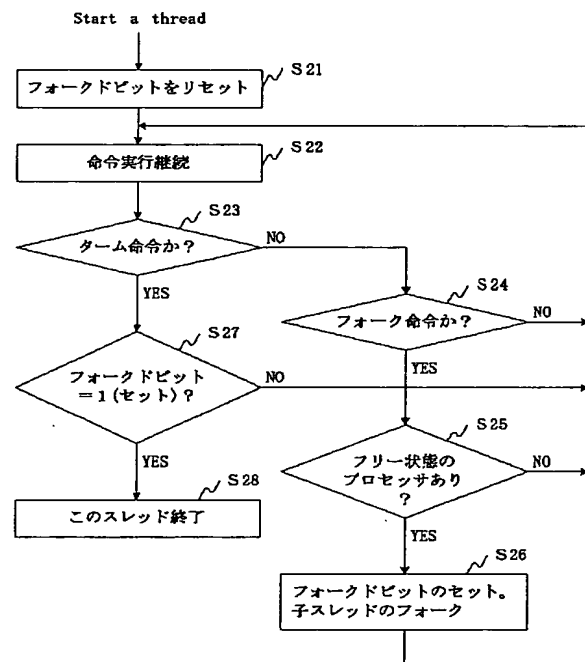
【図5】

【図5】



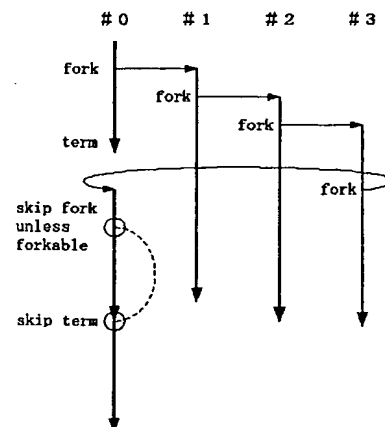
【図7】

【図7】



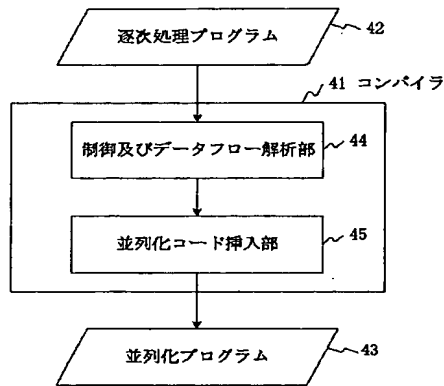
【図8】

【図8】



【図 9】

【図 9】



【図 10】

【図 10】

```

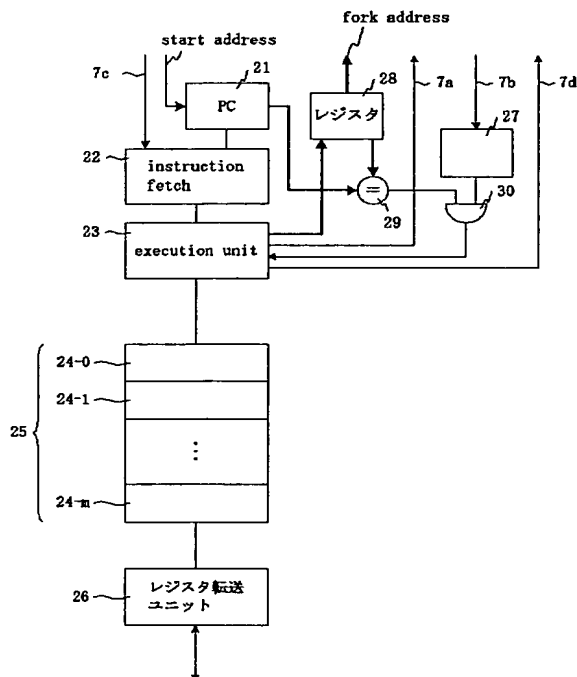
(a)
    addu    r14, r14, 5
    sw      r14, (r16)
    lw      r1, 4(r16)
    _loop:  jsl      _func
            sub     r1, r1, 1
            bne    r1, r0, _loop

    fork    _th1
    addu    r14, r14, 5
    sw      r14, (r16)
    term

(b)
    _th1:   lw      r1, 4(r16)
    _loop:  fork    _th2
            jsl     _func
            term
    _th2:   sub     r1, r1, 1
            bne    r1, r0, _loop
  
```

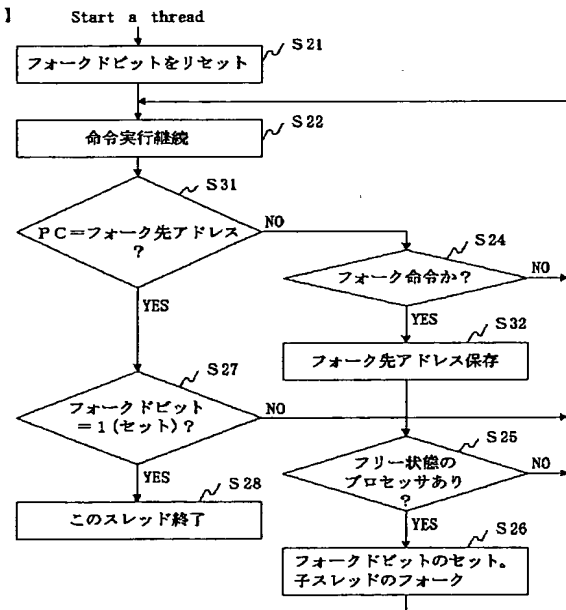
【図 11】

【図 11】



【図 12】

【図 12】



【図 13】

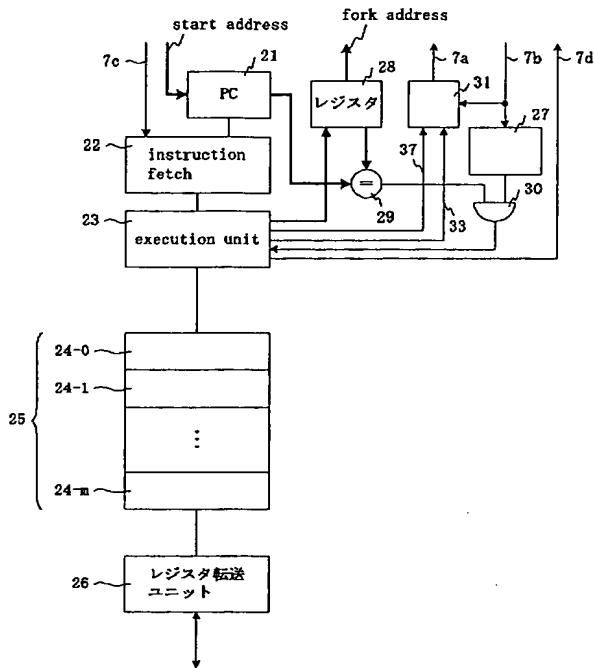
【図 13】

```

fork    _th1
addu    r14, r14, 5
sw      r14, (r16)
_th1:   lw      r1, 4(r16)
_loop:  fork    _th2
        jsl     _func
        _th2:   sub     r1, r1, 1
        bne    r1, r0, _loop
  
```

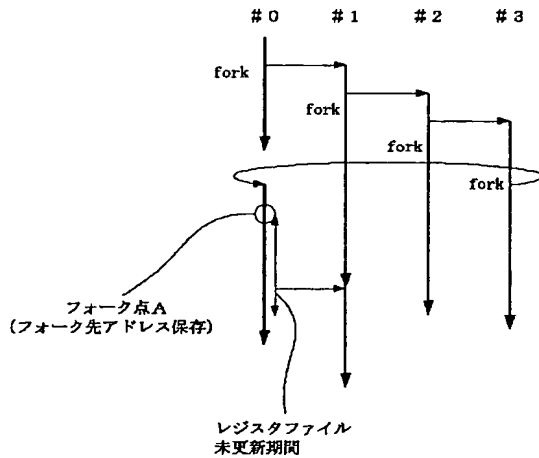
【図14】

【図14】



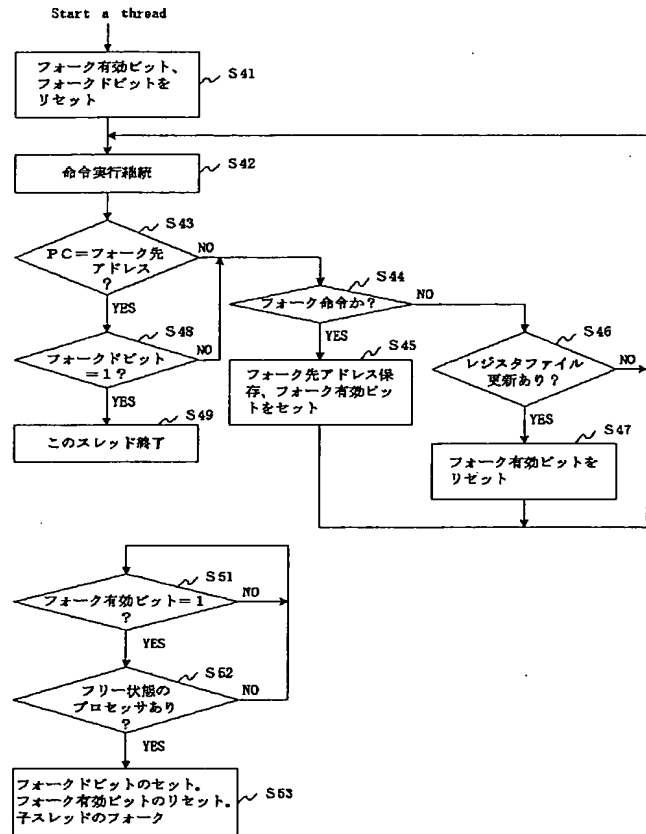
【図16】

【図16】



【図15】

【図15】



【図19】

【図19】

```

fork      .th1, r1, r16, sp
addu     r14, r14, 5
sw       r14, (r16)
.th1:    lw      r1, 4(r16)
.loop:   fork    .th2, r1, sp
jal      .func
.th2:    sub     r1, r1, 1
bne     r1, r0, .loop

```

【図20】

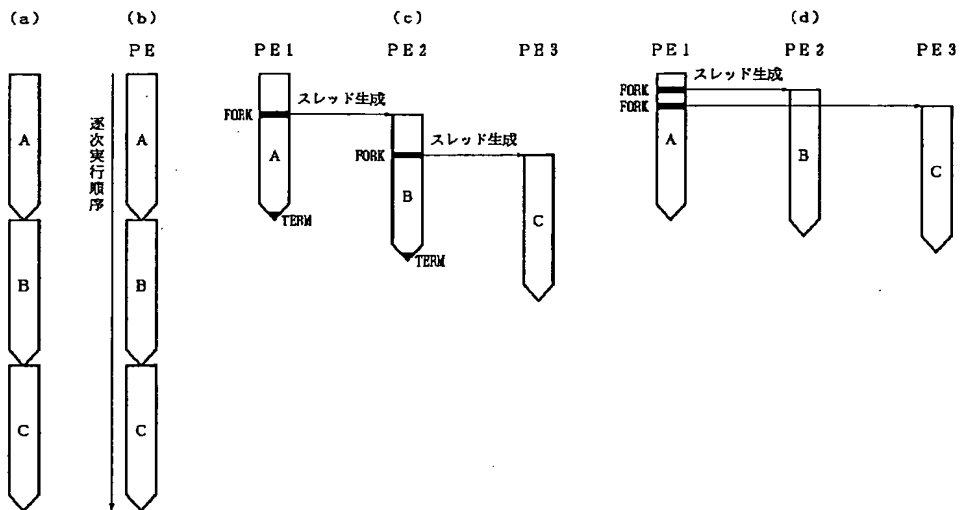
【図20】

```

fork      .th1,
addu     r14, r14, 5
sw       r14, (r16)
.th1:    lw      r1, 4(r16)
.loop:   create  r1, sp
fork     .th2
jal      .func
.th2:    sub     r1, r1, 1
bne     r1, r0, .loop

```


【図 22】



【図 22】

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ **BLACK BORDERS**
- ☐ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- ☐ **FADED TEXT OR DRAWING**
- ☐ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- ☐ **SKEWED/SLANTED IMAGES**
- ☐ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- ☐ **GRAY SCALE DOCUMENTS**
- ☒ **LINES OR MARKS ON ORIGINAL DOCUMENT**
- ☐ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- ☐ **OTHER:** _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.